University Mohamed Khider of Biskra
Domain: Mathematics/Computer Science

Course handout: for 1st year students in Mathematics

Algorithms and Data Structures 1

Dr. Soheyb Ayad

Academic Year: 2024-2025

The content

Chapter 1: Introduction to Computer Science

1. Definitions

1.1. Computer science

Computer science defines all the sciences and techniques related to the automatic processing of information (create, delete, store, modify, process, and present data).

1.2. A computer

Is a machine composed of a collection of entities (hardware, software), designed for the automatic processing of information. It can process various types of information (texts, images, videos, sounds …) but internally all this information is converted into binary digital form (0 and 1).

2. A brief history about the evolution of Computer Science

Pascaline (1642), mechanical calculator, invented by Blaise Pascal and considered the first (purely mechanical) calculating machine.

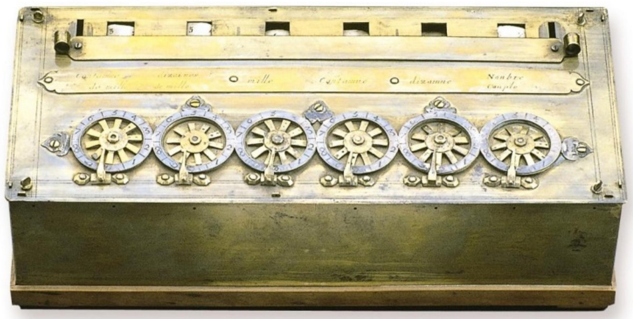


Fig. Pascaline calculator.

The Jacquard loom (1801) is a loom developed by the Frenchman Joseph Marie Jacquard, the first programmable mechanical system with punched cards.

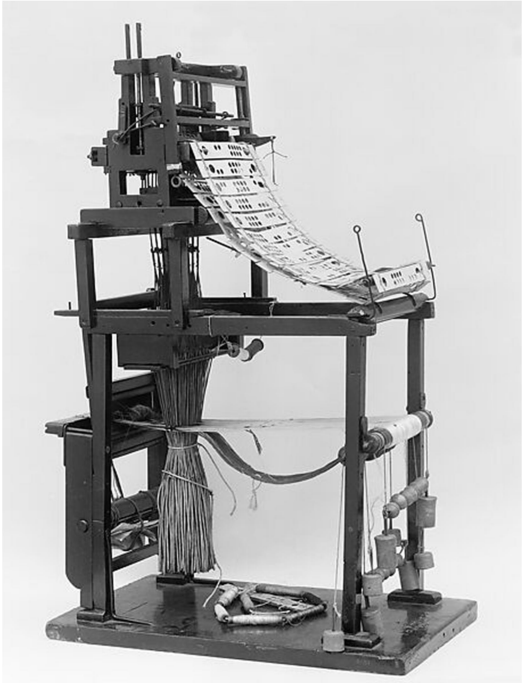


Fig. The Jacquard loom

The analytical machine (1834) is a programmable calculating machine designed by the English mathematician Charles Babbage and which is considered the first true precursor of computing. His "analytical machine", which uses punched cards (The Jacquard loom) indicating the data and operations to be carried out (similar to the program), was theoretically innovative but technically so complex that it never worked.

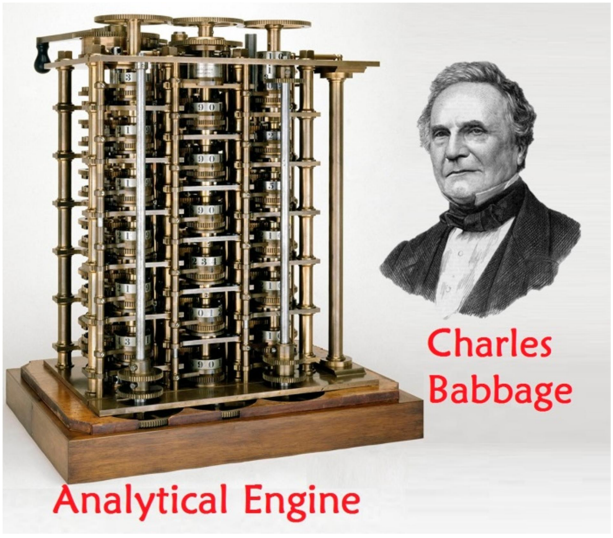


Fig. The analytical machine

It was during the development of Chargle Babbage's machine that **Ada Lovelace** developed the first programming algorithm in history, becoming humanity's first computer scientist.

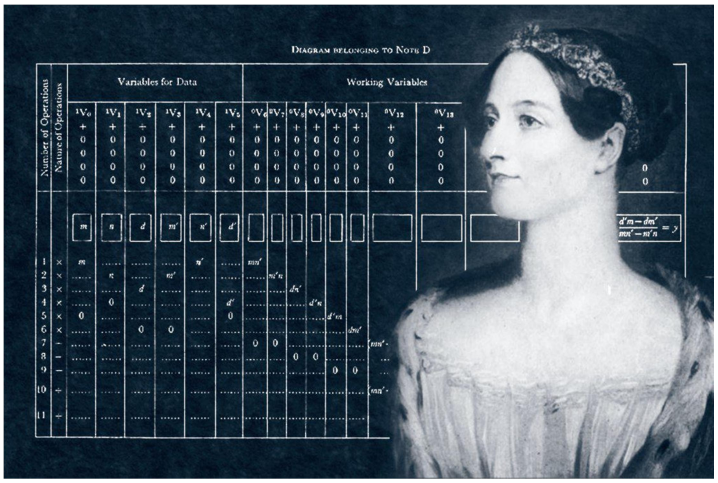


Fig. Ada Lovelace the first computer programmer

The Turing machine (1936), which is one of the basic concepts of computer science, founded by Alan Turing (British mathematician), is a pure abstract mathematical object, which formalizes the operation of mechanical calculation devices.

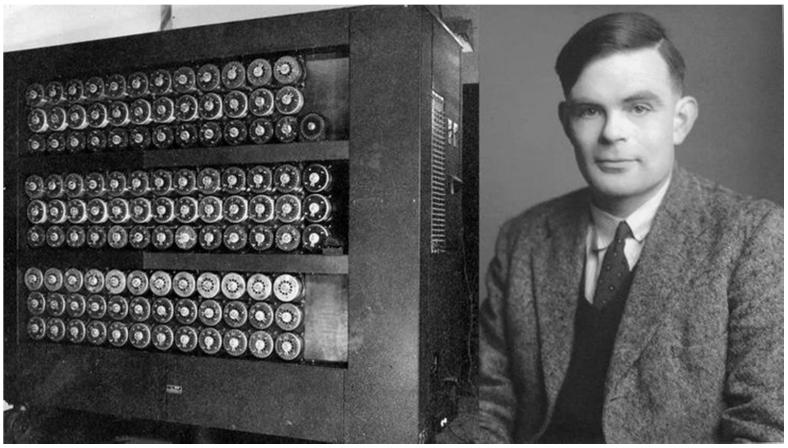


Fig. The analytical machine

Von Neumann (1944), designed the Eniac (Electronic Numerical Integrator And Computer), the first large entirely electronic calculator. Von Neumann is the designer of the architecture of classical computers.

It was in Great Britain, in 1948, at the University of Manchester, that the first computer was finally built. Alan Turing participated in its technical development and it was he who wrote the first "language" to communicate information to the machine. This very simple language is made up of around fifty instructions and it is the machine which translates them into binary language.

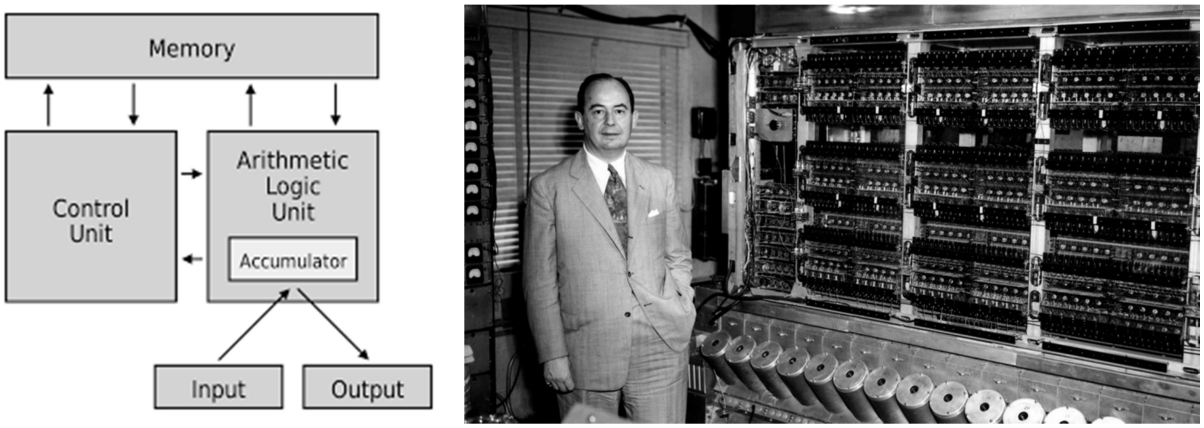


Fig. ENIAC (based on Von Neumann architecture)

IBM-360

In 1964, the American firm IBM marketed the IBM-360 computer. This computer was equipped with printed circuits and was a real commercial success. Computing was no longer just a tool mainly used for military applications or scientific research; it was penetrating massively into businesses.

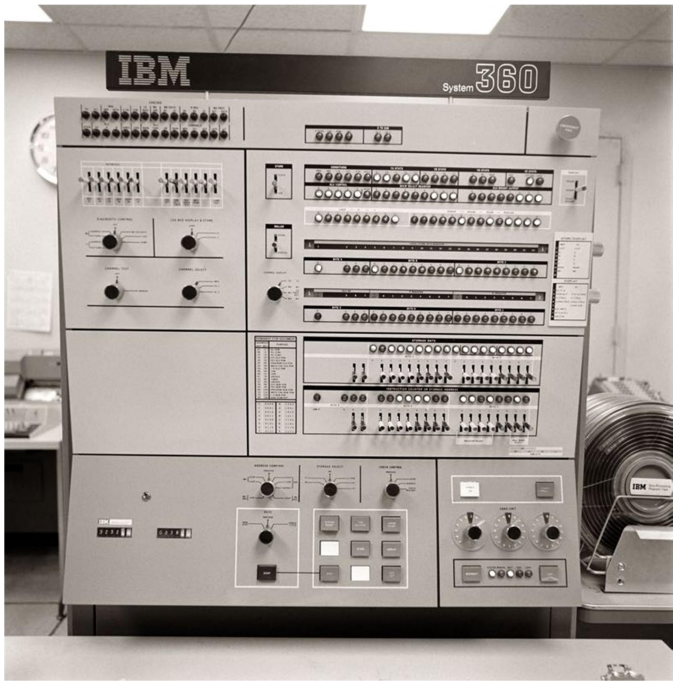


Fig. IBM-360

The microcomputer

Developed from 1975. The immense progress in the miniaturization of circuits led to the invention of the microprocessor, which is the basic organ of the microcomputer.

The first microcomputers were developed and marketed by students. Apple founders (in a garage!).

In 1983 The Lisa machine (Lisa OS) (very slow and expensive)

In 1984, Apple developed the Macintosh (using a mouse inspired by Xerox laboratories)

The success is worldwide and will inspire Bill Gates for his Windows system (Microsoft).

Bill Gates' Microsoft company was successful from 1981, when its system was chosen by IBM to equip its computers known as PCs (Personal computers).

From 2007, new revolution → smartphones era (Apple iPhone)…etc

3. An algorithm

1.1. Definition 1

An algorithm is a series of instructions ordered in a logical and sequential manner indicating the procedure to follow to solve a problem.

Examples :

- The Euclid algorithm which calculate the p.g.c.d. of two integers.
- Sorting algorithms, which allow, for example, ordering values.

Algorithms that do not necessarily only applied to computing:

- Indication of a path to reach a destination.
- Cooking recipe

1.2. Definition 2

An algorithm can be considered as a machine operating in three stages:

1- Enter the necessary data: The entries (or inputs).

2- Sequentially execute instructions on this data: Processing.

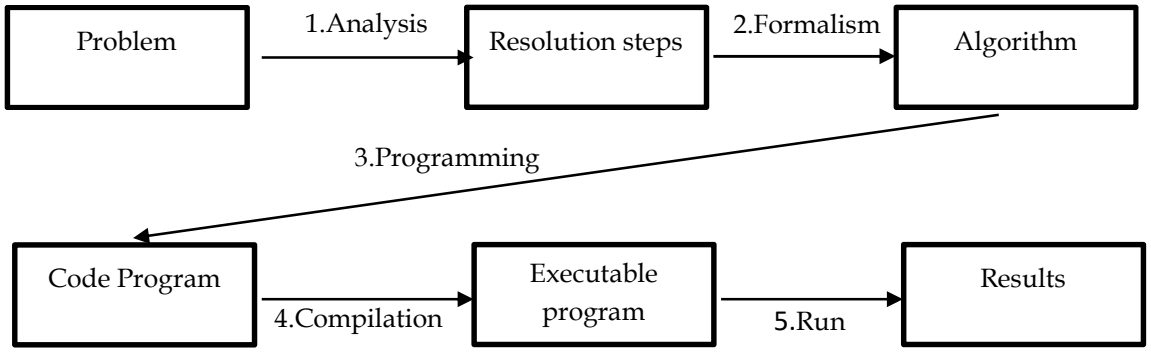
3- Display the obtained results: The outputs.

4. A program

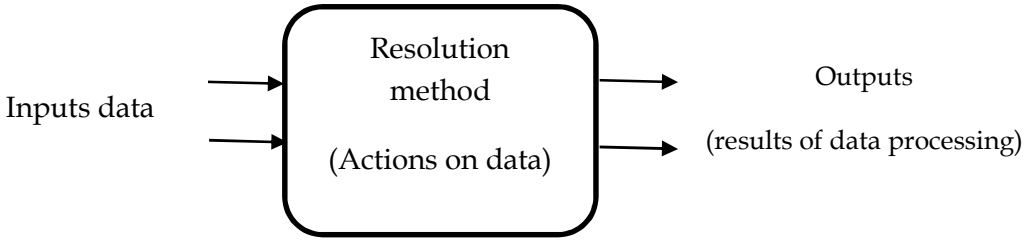
1.1. Definition

A computer program is a logical sequence of instructions (algorithm) written in a well-defined programming language (understandable by machine). Example: C, Pascal, Python, Java, etc.

1.2. Program Construction



- ***The first step:*** consists of analyzing the problem to determine the expected inputs and outputs as well as the elementary operations which describe the resolution method.



- ***The second step:*** is the establishment of an algorithm which is a formalism to represent the steps for solving the problem analyzed.
- ***The third step:*** is the translation of the algorithm into a program using a chosen programming language (C language, for example).
- ***The fourth step:*** Once the program is written, you will have to check it and correct it using the compilation method. The compiler is software that detects program syntax errors, but does not detect semantic errors.
- ***The last step:*** consists of executing the compiled program. This step leads to the exploitation of the program and the verification of the results.

Note:

When writing a program, you may encounter two types of errors:

Syntactic errors: they are noticed during compilation and are the result of poor writing in the programming language.

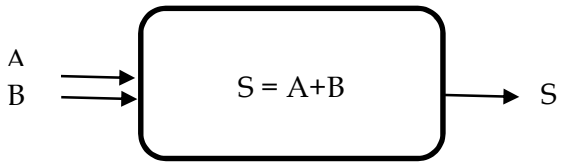
Semantic errors: they are noticed at execution, they are the result of poor analysis. These errors are much more serious because they can appear while the program is running.

Example:

Problem

Write an algorithm that calculate the addition of two real numbers A and B.

Analysis



- Determine the inputs and outputs:
  Inputs A, B: real values
  S = A + B // A and B are real, so, S is also real.

- Determine the methods:

  Read the values of A and B from the keyboard (input).
  Evaluate: S = A + B
  Write into the screen the value of S (result)

Algorithm (formalism)

```
Algorithm Add ;
 Var
   A, B, S : Real ;
 Begin
       read ( A ) ;
       read ( B ) ;
       S ← A + B ;
       write ( S ) ;
 end.
```

Program (code in C language)

```
#include <stdio.h>
int main( )
{
        float A, B, S ;
        scanf("%f", &A) ;
        scanf("%f", &B) ;
        S = A + B ;
        printf("The sum = %f \n", S) ;
        return 0 ;
}
```

Chapter 2: Simple sequential algorithm

1. Concept of algorithmic language

Algorithmic language:
The algorithmic language is a compromise between natural language and a programming language, characterized by a list of keywords.

The key word:
A keyword is an identifier that has a specific meaning to the programming language (Begin, End, If, For, Repeat, While, etc.).

2. Structure of an algorithm

An algorithm can be presented as follows:

Algorithm name_of_algorithm; **// Header**

Const List_of_constants;
Var List_of_variables;
Type List_of_types;
// Declaration

List of procedures/functions;

Begin
Instruction 1 ;
Instruction 2 ;
…
Instruction N ;
end.

// Body of algorithm
// Initialization → processing → results display

✓ **The header:** used to identify the algorithm by giving it a name.
✓ **Declarations**: This is a list of all objects (constants, variables, etc.) used and manipulated in the body of the algorithm.
✓ **The body**: Contains the operations and instructions to be executed.

Example :
The algorithm that calculates and displays the area and perimeter of a circle of radius R can be written as follows:

Analysis

Inputs: R
Ouputs : S and P
R is real so S and P are reals too.

Operations

1. Read the value of the radius R
2. Calculate the surface S according to the law $S = \pi * R^2$
3. Calculate the perimeter P according to the law $P = 2* \pi * R$
4. Present (display) the values of: S and P

Formalism

Algorithm cercle ;
Const
Pi = 3,14 ;
Var
R, S, P : Real ;
Begin
Read (R) ;
S ← Pi x R x R;
P ← 2 x Pi x R;
Write (S) ;
Write (P) ;
end.

3. Variables and constants

3.1. Definition of a variable

A variable is a memory space identified by a name (address), used to store a value, which can be modified during the execution of the algorithm.

3.2. Definition of a constant

A constant only takes on a single value during the execution of the algorithm.

3.3. Variable declaration

To declare a variable we define its name and its type.

Syntax

Var

Name_Variable1: Type1 ;

Name_Variable2,…, Name_VariableN : Type2 ;

Example

Algorithm	Program (C)
Var A, B : integer ;	int A, B ;
C : real ;	float C ;
X1 ,X2 : boolean ;	bool X1,X2 ;
id_1 : character ;	char id_1 ;

3.4. Constantes declaration

To declare a constant we define its name and its value.

Syntax

Const

Name_constante1 = value1;

Name_constante2 = value2;

Example

Algorithm	Program (C)
Const Pi = 3.14 ;	Const Float Pi=3.14 ;
resp = true ;	bool resp = true ;

A variable is identified by an **identifier** and a **type** and a **value**.

A. The identifier

An identifier is the name of a variable, a constant or the name of an algorithm (program). An identifier must respect a particular syntax:

- It is made up by a serie of letters, alphabets (in upper and lower case) and numbers or the underlined character "_" (the underscore).
- It must begin with a letter or "_". (The name must not start with a number).
- An identifier cannot be a keyword (Start, End, If, Then, Else, printf, if, else, while, for…..etc).
- An identifier must not contain special characters like: ?, !, *, …

Remarks :

- To facilitate the readability of an algorithm, it is preferable to use meaningful names.
- Characters of the Greek alphabet are prohibited. If we need to use α, β, or δ, we must write them in Latin alphabet like alpha, beta, delta and so on.
- It is forbidden to use accented characters (é, è, î, ç, ', ", …) in an identifier.
- It is forbidden to use indices or exponents (do not use x_1, x^2 but rather use x1 and x2)
- The C language is case sensitive (The variable Age ≠ age)
- *Examples*: x, y, PI, radius_of_the_circle, _x1, _y2, …

B. The type

A variable is assigned a type. This type indicates the memory space that can be reserved for this variable.

The standard (basic) types in algorithmic language are: Integer, Real, Boolean, Character.

- **The integer type:** designates a set of integer values. Examples: -10, -6, 0, 5, 1001...
- **The real type:** designates the set of real values. Examples: 1.55, -224.25 ...
- **The boolean type:** designates two values: TRUE or FALSE.
- **The character type:** This type includes all the characters used for writing text ('a', 'b 'c', 'A', 'B', '*', '+', ';'. ..).

Remarks :

- There are also composed types defined from basic types such as arrays, records, strings, etc. Also, we can define a new type and give it a new name.
- The representation of variables in memory depends on the systems and languages used.

C. The value

The value can change during the algorithm (program), but must respect the type. A variable whose type is integer can therefore never contain a value with comma.

4. Basic operations

4.1. Expressions

An algorithm can contain expressions.
An expression is a combination of operands (variables or values) and operators

Exp:

X operand + operator Y operand

An OPERATOR: is a tool that allows performing calculations.
An OPERAND: can be a value, a constant, a variable or a function call which will be used by an operator.

4.2. Types of operators

A. Arithmetic operators

Addition +, Subtraction -, Division /, Multiplication *, Equal =, MOD and DIV

MOD : give the remainder of division (x MOD y).
DIV : give the quotient of division (x DIV y).

Algorithm	Program C
DIV	/
MOD	%

B. Comparaison operators

Strictement Inférieur **<**, Inférieur ou égale ≤, Strictement Supérieur **>**, Supérieur ou égale ≥, Différent ≠, Strictly inferior <, inferior or equal than ≤, Strictly superior >, superior or equal than ≥, Different ≠

Algorithm	Program C
≤	>=
≥	<=
=	==
≠	!=

Note: The result of a comparison will be TRUE (1)/FALSE (0)

C. Logical operations

AND : logical AND, **OR :** logical OR, **NOT** : negation

Algorithm	Program C
AND	&&
OR	||
NOT	!

Example: Logical table

A	B	A AND B	A OR B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

A	NOT A
False	True
True	False

4.3. The rules for evaluating an expression

Calculating the value of an expression containing more than one operator depends on the meaning given to this expression. Example: X + Y * Z is an ambiguous expression.

Parentheses remove ambiguity: (X + Y) * Z or X + (Y * Z).

To avoid any ambiguity in the absence of parentheses, evaluation rules have been established. An order of priority between the operators is introduced as follows:

Level 1 : ( ) // ---- highest priority
Level 2 : NOT
Level 3 : * , / , MOD , DIV , AND
Level 4 : + , - , OR
Level 5 : = , < , >, ≥, ≤ // ---- least priority

Remarks:

As best practice in writing algorithms

- We must always add parentheses in the expressions to avoid ambiguities.
- The algorithmic expressions must be written in linear form.

Examples :

$\frac{X+Y}{2Z}$ → (X+Y)/ (2*Z) ;

B^2 - 4 AC → (B * B) – (4 * A * C)

NOT A AND B → NOT (A AND B) // (NOT A) AND B

A ∈ [0,255] → (A >=0) AND (A<=255)

4.4. Basic (elementary) instructions

In general, there are three elementary instructions: Read, Write and assignment ←

A. Assignments

The assignment is an instruction that stores the value of an expression in a variable.

Syntax

VariableName ← expression

Examples

```
X ← 7                  // Assign a value to the variable X (X receives 7)
Y ← X                  // Copy the value of variable X into variable Y (Y receives X)
Z ← (2 * X + T) / Y    // An expression formed from several operations ( Z receives (2 * X + T) / Y )
```

Remarks

- If the variable already contained a value, it would be replaced by the value of the expression. The old value of the variable will be lost and there is no way to recover it!

- A type check operation is performed before assignment. It is an error if the value of the expression does not belong to the type of the variable, unless it is an integer value and the variable is of real type. In this case the integer value is converted to real.

Exercises

Exercise 1

What will be the values of variables a and b after performing the following actions?

```
Algorithm Ex1 ;
 var
    a,b : integer ;
 Begin
    a ← 1 ;
    b ← a + 3 ;
    a ← 3 ;
 end.
```

Exercise 2

What will be the values of variables a, b and c after executing the following instructions?

```
Algorithm Ex2;
  var
    a, b, c : integer ;
 Begin
   a ← 5 ;
   a ← a – 1 ;
   b ← 3 ;
   c ← a +  b ;
   a ← 2 ;
   c ← b – a ;
   c ← c +1 ;
end.
```

B. Input-output operations

The reading instruction (READ)

The READ instruction allows assigning a value to a variable from the keyboard.

Syntax: read(VariableName);

Examples:

```
read(A);
read(B);
```

- Several successive reading instructions can be grouped into a single instruction. For the previous example, we can replace the three reading instructions with:
  read(A, B).
- On a computer, when the processor receives the read(VariableName) order, it stops execution of the program and waits for a value. The user must then enter a value from the keyboard. As soon as the entry is validated (by pressing the Enter key ↵), execution of the program continues. The value passed by the user is assigned to the variable and overwrites its previous value.
- The read(VariableName) instruction causes an error if the value entered does not belong to the type of the variable, unless it is an integer value and the variable is of real type. In this case the integer value is converted to a real value.

The writing instruction (write)

The write instruction allows display on screen. There are two types of display:

1. Display the value of a variable or an expression.

   Syntax: Write (VariableName);

   Example : Write(X) ; or Write (2 * X + Y) ;

// If X = 10 and Y= 5 then, the first instruction display 10 and the second display 25 .

2. Display a text message.

   Syntax : Write ("a message") ;
   Example : Write(" The result is = ") ;

Different successive write instructions can be grouped into a single instruction. For example, the sequence of instructions:

Write ("The result is =");
Write (X) ;
Can be replaced by: Write ("The result is =", X);

This instruction displays : "The result is = 10".

Note: Message communication is very useful in programming. It offers the possibility:

- To guide the user by telling him what the machine expects from him following a reading order.
- To explain the results of a treatment.

C. Comments

A comment is optional text that has no effect on the instructions of the algorithm. It is used to explain why certain instructions are used in a program.

Syntax of a comment:

/* comment */ or // comment

Example :

```
Algorithm Sum;
 var
   A, B, C: integer;
 begin
    /* this program calculates the sum of two numbers*/
   Read (A);
   Read (B);
   C ← A+B;  //The variable C will contain the result
   Write (''the sum ='', C);
 end.
```

D. Inputs and Outputs in C language

Syntax :

Algorithm	Program C
←	=
read (A)	scanf('' %d'', &A) ;
write (A)	printf(''%d'', A) ;

E. Variable content format

The following table represents the main formats that can be used with input-output functions in C language:

Format	Description
%d	Data with Integer type
%f	Data with real type
%c	Data with character type
%s	Data with string type

F. Exercises

Exercise 1

Write an algorithm that calculates the perimeter and area of a rectangle. Translate into C language.

Exercise 2

Write an algorithm that calculates the sum and average of 5 exam notes.

Exercise 3

Write an algorithm that calculates the sum and average of 5 exam scores, using only two variables.

Exercise 4

Write an algorithm that reads two integer values A, B, then swap them and display the result.

exp: if A=12 and B = 23 after swapping A= 23 and B = 12

Translate into C language.

5. Flowchart representation

A flowchart is a diagram that illustrates the steps and decisions of a process. It can be used to graphically represent an algorithm.

The flowchart contains standard forms. Each form describes a specific operation.

The basic forms of a flowchart are represented in table below:

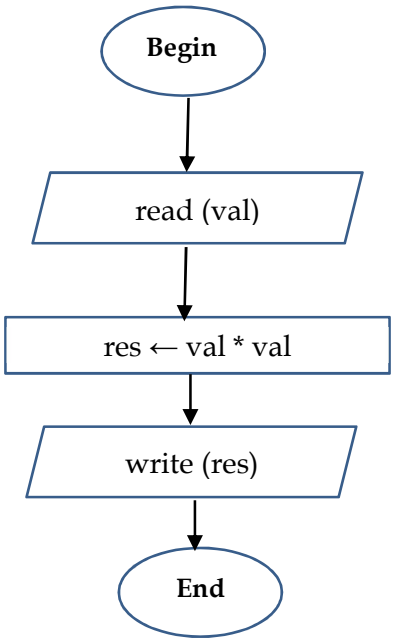
Form	Description
	Start(Begin)/end of the flowchart
	Processing (operation)
	Write, Read, Input, Output operations
	A subroutine (subprogram) : Call to a portion of an algorithm (program) considered as a simple operation
	Decision (based on condition)
	Flow lines : show the direction of the flow (process).

Example :

Consider the following algorithm which calculates the square power of a number Val.

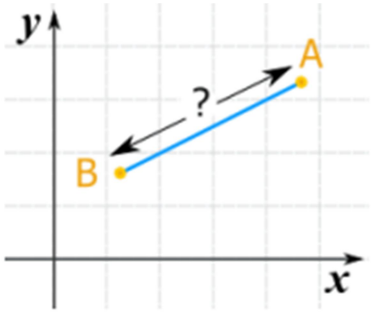
Algorithm square_pow;
Var
    val, res : integer;
Begin
    read(val);
    res ← val * val ;
    Write ("the result is: ", res);
End.

The corresponding flowchart is represented as follows:



Exercise:

1- Write an algorithm that calculates the distance between two points A and B in a plane (X, Y).



$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

2- Represent the algorithm using a flowchart.

3- Translate the algorithm to a program using C language.

Chapter 3: Control structures

1. Introduction

The control structures allow an algorithm (program) to not be purely sequential (linear chain of instructions).

In certain cases, the algorithm may:

- only execute specific instructions under certain conditions.
- repeat the same instructions several times.

In general, there are two types of control structures that allow this.

- The conditional structures: only execute certain instructions under certain conditions.
- The repetitive structures (loops): repeat the same instructions a certain number of times (under certain conditions)

2. Conditional structures

There are three types of conditional structures:

- Simple conditional structure: if. . .endif
- Compound conditional structure: if. . . . else . . . . endif.
- Multiple choice structure: AccordingTo. . else. . endAccordingTo.

2.1. Simple conditional structure

This structure allows to execute one or more instructions if a condition (a condition is an expression whose evaluation result can be TRUE or FALSE) is true and to exit the control structure if the condition is false.

The expression can be simple (single condition) or composed (several composed conditions with logical operators AND, OR, NOT).

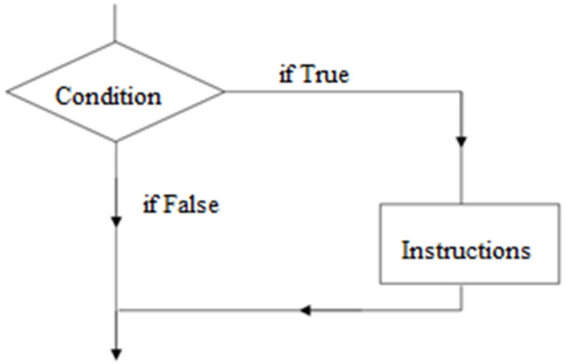
Syntax:

if (condition) **then**

Instructions ;

Endif ;

The simple conditional structure can be represented in a flowchart as follows:



A. Examples :

Exp 1 :

```
Algorithm positive;
Var
  X : integer ;
Begin
 read(X) ;
  if (X ≥ 0) then
    write(X) ;
  Endif ;
End.
```

Semantics: *We display the value of X if it is only greater than or equal to 0.*

Exp 2 :

```
Algorithm positive ;
Var
  X : integer ;
Begin
   Read(X) ;
   if (X < 0) then
     X ← X * ( -1);
   Endif ;
 Write(X) ;
End.
```

Semantics: *If the value of X is negative (less than 0) we multiply it by -1, the displayed result is always a positive value.*

Exp 3 :

```
Algorithm range ;
Var
  X : integer ;
Begin
  read(X) ;
  if ((X > 10) AND (X < 15)) then
    Write(X) ;
  Endif ;
End.
```

Semantics: *We display the value of X if it is between 10 and 15.*

Exp 4 :

```
Algorithm range ;
Var
  X : integer ;
Begin
 Read(X) ;
 if (X > 10) then
    if (X < 15) then
         write(X) ;
    endif ;
  endif;
end.
```

Semantics: *This example is equivalent to the previous example, except that this time we use nested tests.*

B. Simple conditional structure using C language:

The simple conditional structure is represented in C language as follows:

Algorithm	Program in C

if (condition) **then**
Instructions ;
endif ;
	if (condition)
	{
	Instructions ;
	}

Remark : If the bloc of instructions is composed of only one operation the brace symbol "{ }" become not obligatory.

Example :

Algorithm	**Program 1**	**Program 2**

Algorithm condition ;
Var
X : integer ;
Begin
read(X) ;
if (X > 10) then
write(X) ;
endif ;
end.
	int main ()
	{
	Int X;
	Scanf(''%d'' , &X);
	if (X > 10)
	{
	printf(X) ;
	}
	
	return 0 ;
	}
		Int main ()
		{
		Int X;
		Scanf(''%d'' , &X);
		if (X > 10)
		printf(X) ;
		
		return 0 ;
		}

Programs 1 and 2 are both correct.

Exercises :

1) Represent the examples 3 and 4 by a flowchart.
2) Study the flowing algorithms.

Exp a :	***Exp b :***

Begin
Read(X) ;
if (X < 10) then
if (X > 15) then
write(X) ;
endif ;
endif ;
end.
	Begin
	read(X) ;
	if (X < 10) OR (X > 15) then
	write(X) ;
	endif ;
	end.

Semantics :	***Semantics : ... .***

2.2. Compound conditional structure

Allows choosing between two blocks of instructions which ones to execute, according to the condition, if it is true, the first block of instructions is executed, otherwise, if it is false, the second block of instructions is executed.

Syntax:

if (condition) **then**

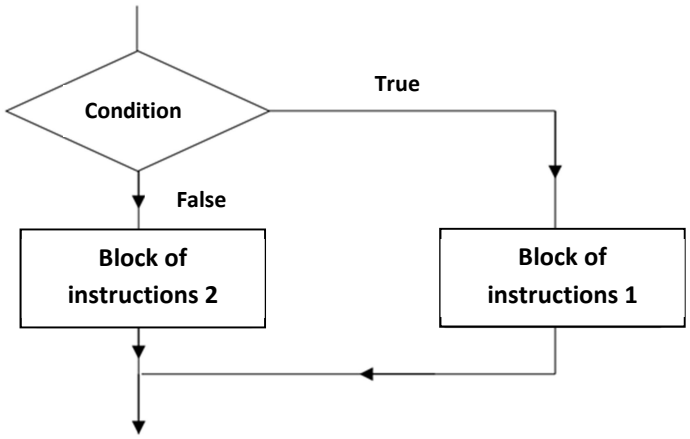
Block of instructions 1;

else

Block of instructions 2;

endif ;

The simple conditional structure can be represented in a flowchart as follows:



A. Translation to C programming language

The compound control structure is represented in C language as follows:

Algorithm	Program in C

if (condition) then
Block of instructions 1;
else
Block of instructions 2;
endif ;
	If (condition)
	{
	Block of instructions 1;
	}
	else
	{
	Block of instructions 2;
	}

Example:

Consider the following algorithm which displays whether a number is positive or negative.

```
Algorithm pos_neg;
var
    A: integer;
Begin
write ("Enter the value :");
read (A);
  if (A ≥ 0) then
    write (“positive”);
  else
    write (“negative”);
  endif ;
End.
```

Question: Translate the algorithm into C program.

Exercises:

1- Write an algorithm that displays the price of a printers order. The price of single printer (Canon 3110) is 50000 DA. From the fifth printer any other purchased will be counted 45000 DA. Translate the algorithm into C program.

Solution:

```
Algorithm order ;
  Var
    Nb, price : integer ;
 Begin
   read (nb);
   if (nb > 5) then
      price ←  (50000*5)+(45000 x (nb-5 )) ;
  else
      price ←  5000 x nb ;
  endif ;
  write(''the total price = '', price) ;
Fin.
```

2- Write an algorithm that displays the set of solutions on R of the first degree equation ax + b = 0

Solution:

```
Algorithm  first_degree;
Var
      a,b:integer;
Begin
   Read(a,b);
   If (a≠0) Then
        Write("The solution of the equation = ",-b/a);
   else
     if (b=0) Then
          Write("The solution of the equation is indeterminate");
    else
          Write("The solution to the equation is impossible");
    endif ;
   endif ;
end.
```

2.3. Multiple choice conditional structures

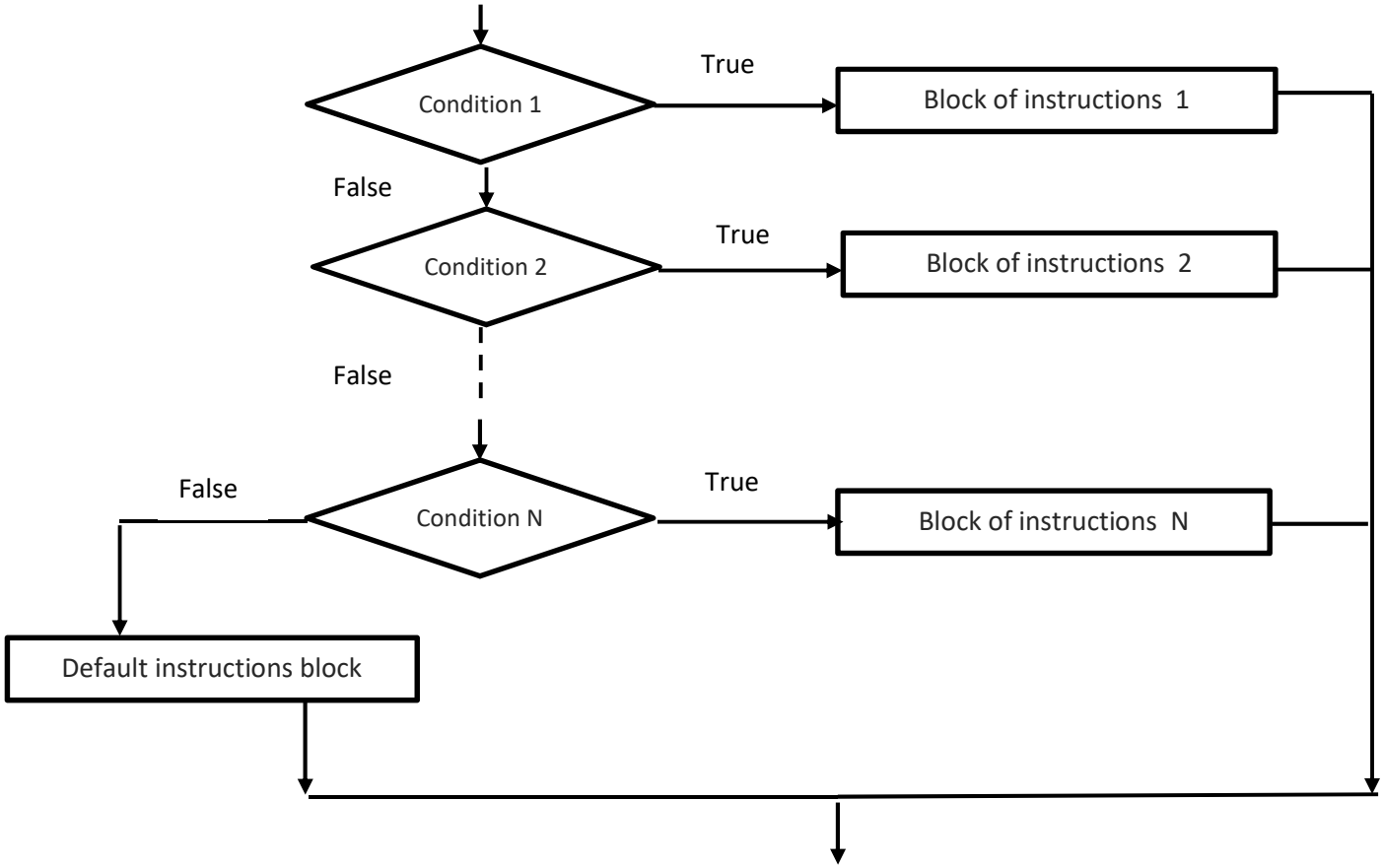
The multiple choice structure allows comparing an object (variable or expression) to a set of values, and executes one of several blocks of instructions, depending on the actual value of the object. A default statement block may be provided in the case where the object is not equal to any of the listed values.

This structure avoids the need for nested conditional structures and offers better comprehension of the solution.

Syntax :

Algorithm	Program in C
AccordingTo (variable ou expression) do Value 1 : block of instructions 1 ; Value 2 : block of instructions 2 ; …… Value N : block of instructions N ; **Else** default instructions block; **endAccordingTo;**	**switch(x)** **{** case value_1 : block of instructions 1 ; Break; case value_2 : block of instructions 2 ; Break; …….. case value_N : block of instructions N ; Break; **default:** default instructions block; **}**

The multiple choice conditional structure can be represented in a flowchart as follows:



Remarks:

- The default line is optional
- In the multiple choice instruction, the order of presentation changes nothing.
- The expression and the cases values must be in the same type.

Example:

In the following example, the value of X is displayed in letters, if it is equal to 1, 2 or 3 otherwise it displays an error message.

```
Algorithm display_letter;
  Var
    X: integer;
  Begin
    Read(X);
    AccordingTo (X) do
      1: write (“One”);
      2: write (“Two”);
      3: write (“Three”);
      Else  write(‘’The value entered is: > 3 or < 1’’);
     endAccordingTo;
 end.
```

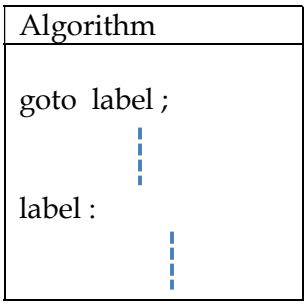
Program in C :

```
#include <stdio.h>
Int main ()
{
  Int x;
  Scanf(‘’%d’’,&x);
  switch(x)
  {
    case 1 : printf(“one”);
             Break;
    case 2 : printf(“two”);
             Break;
    case 3 : printf(“Three”);
             Break;
   default:
             printf(“the value entered is : > 3 or < 1”);
  }
```

2.4. Branching operator

It is possible to jump directly to an instruction using a branch operation of the form **goto identifier**. The identifiers are labels or addresses of branching.

Syntax:



Example :

```
Algorithm branching;
  Var
        a: integer;
 begin
        read (a);
        goto 1;
        a ← a+1 ;
        1: a ← a+2;
        write (a) ;
 End.
```

If the value of a = 5 the displayed result will be 7.

Note :

The use of the go to branch should be avoided, because it reduces the readability of the codes and often leads to semantic errors.

Exercise: Consider the following algorithm:

```
Algorithm Branching;
  Var
     a : integer;
 begin
    label1: write (" Enter an integer number :");
    read (a);
    if (a mod 2 = 0) then
       goto label2; // conditional branching
     endif ;
    write("the number entred is odd !");
    goto label1; // unconditional branching
    label2: write ("the number is even");
 end.
```

Write the algorithm in C language ?

Solution :

```
#include<stdio.h>
void main()
{
int a,b;
label1: printf (" Enter an integer number :"):
scanf("%d",&a);
if (a % 2 == 0)
 {
   goto label2 ;
 }
  printf("\n the number entred is odd  ! \n ");
  goto label1 ;
  label2 : printf("\n the number is even \n ");
}
```

Exercise 1:

Using the branch instruction, write an algorithm that calculates the sum of two real numbers entered on the keyboard at the request of the user, it means, repeat the steps by asking the user at the end of each calculation whether he wishes to repeat the sum calculation operation (Yes/No) or not.

- Translate into a C program.

Solution in C language :

```
#include<stdio.h>
void main()
{
        int a,b,s,c;

        et1:
           scanf("%d",&a);
           scanf("%d",&b);
           s=a+b;
           printf("la somme = %d \n",s);
           printf("do you want to repeat the calculation operation ?  1 : Yes, 2 : No");
           scanf("%d", &c);
           if(c==1)
                 goto et1;

           printf("\n Thank you \n");
}
```

Exercise 2 :

Write an algorithm that asks the user for two integers and, without calculating the product, it displays whether the product is negative, positive or equal to zero.

- Translate into a C program.

Chapirte 4 : Repetitive structures « loops »

1. Introduction

In algorithms, some instructions may be repeated and it would not be very wise to rewrite them. Repetitive structures can be used to control repetition.

A repetitive structure (or iterative structure) repeats the execution of instructions in a determined or indeterminate number of times. An iterative structure is also called a loop.

Mainly, there are three forms of loops, the use of one or the other is related to the repetition test and algorithm context.

2. The loop "While"

The loop While allows you to repeat the same block of instructions several times as long as a condition remains true.

Syntax:

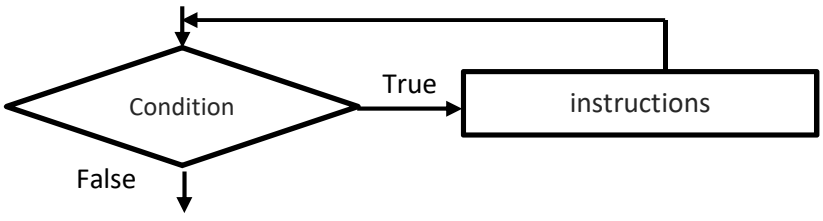
while (Condition**) do**
  Instruction 1 ;
  Instruction 2 ;
  ……
  Instruction 3 ;
endwhile;

// Instructions to repeat;

Semantic :

If the condition is true the group of instructions is executed then **endwhile** returns execution to the condition test, and so on until the condition will be evaluates to false.

Flowchart representation:



The While loop in C language:

While (Condition**)**
{
instructions ;
}

Remarks :

- The block of instructions being repeated until the condition becomes false, means that the instructions to be repeated must modify the parameters involved in the condition in order to exit the loop. If the condition always remains true we can get an infinite loop problem.

Example: The following algorithm calculates the sum of integers in the interval [1, 30]

Algorithm Sum;
Var
i, s : integer;
Begin
s ← 0 ;
i ← 1 ;
While (i ≤ 30) **Do**
s ← s + i;
i ← i+1;
EndWhile;
Write (''the sum is : '', s);
end.

<u>Application exercises:</u>

Exercise 1: Write an algorithm that asks the user for an integer, and then calculates its factorial.

Exercise 2: Write an algorithm that displays whether an integer A is perfect or not.

Exercise 3: Write an algorithm that calculates the division of a real number by an integer according to the formula: (z = x/y) (y ≠ 0).

Exercise 4: write an algorithm that calculates the number of integer values entered to arrive at a sum not exceeding 500.

3. The loop '' Repeat''

The Repeat loop has the same function of the While loop, except that the condition is at the end of the loop and not at the beginning.

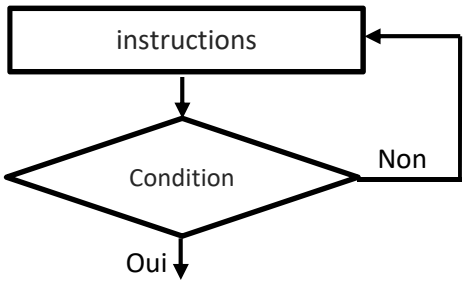
The difference between the two forms is that a loop with While may never be executed because the condition may be false initially while in the loop Repeat the instructions to repeat are executed at least once.

Syntax :

Repeat
  Instruction 1;
  Instruction 2;
  …
  Instruction n ;
Until (Condition);

Note :
The Repeat condition is an end condition where the repetition is finished when the condition is true, whereas, the While condition is a repeat condition where the repetition is finished when the condition evaluates to false.

Flowchart representation:



Example:
Write an algorithm that calculates the division of a real number by an integer according to the formula: (z = x/y) (y ≠ 0).

Solution :

```
Algorithm Division;
 var
            x, z : real;
            y : integer;
 begin
            read (x);
            repeat
              read (y) ;
            until (y ≠ 0);
            z ← x / y;
            write (z);
 end.
```

Analysis :
When executing this algorithm, the transition to the z calculation instruction is only made when the entry of the value of y is different from zero.

Solution using while loop:

```
Algorithm Division;
 var
            x, z : real;
            y : integer;
 begin
            read (x, y);
            while (y = 0) do
              read (y) ;
            endwhile;
            z ← x / y;
            write (z);
 Fin.
```

Analysis:
The read instruction is then written twice: the first to test the condition and the second to repeat the reading until a non-zero value is entered.

Syntax in C language:

```
Do
 {
  Statement 1;
  Instruction 2;
  …
  Statement n;
 }
While( ! Condition);
```

Exercise :
Write an algorithm that assists the user with messages asking for a number between 1 and 10 until the answer matches.

Solution :

```
Algorithm  interval_match;
 var
   n: integer;
 Begin
   Repeat
      Write("Give an integer between 1 and 10");
      Read(n);
      If (n<1 or n>10) Then
         Write("Wrong entry. Please start again!!") ;
      endif;
    until (n>=1 and n<=10);

   Write("The value entered is correct, Thank you!!") ;
end.
```

4. The loop "For"

The loop "For" uses a counter to control the number of iterations (number of repetitions) of the sequence of instructions.

Syntax:
For (counter ← Initial_value **To** Final_value **Step** = n) **Do**
  Instruction block;
EndFor ;

Semantics:
The ***counter*** of iterations is initialized by ***initial_value*** which will automatically increments according to the value the ***Step n*** after each execution (iteration) until reaching the ***final_value***.

Remarks :

- The type of the counter is integer
- The step is optional. It represents the increment step when it is positive and decrement step when it is negative.
- By default the increment step is equal to 1, in this case we do not have to mention it.

Syntax in C language:

```
For(counter=initial_value; Stop condition; Step)
 {
  Instructions block;
}
```

Note:
Stop condition: condition to exit the loop, in general (counter < final_value)
Step: increment or decrement instruction (exp: i=i +1 or i++ / i = i-1 or i--)

```
For (i ← 1 To 10) Do
 write( i ) ;
endfor ;
```

Example1:
To display the numbers from 1 to 10 on the screen, you can use the For loop as follows:

Algorithm	Program in C

For (i ← 1 T**o** 10) **Do**
  Write(i);
EndFor ;
	For(i=1 ; i<10 ; i++)
	{
	  Printf(''%d'', &i) ;
	}

Example2:
To display even numbers from 1 to 10 on the screen, we can use the following For loop:

For (i ← 2 **To** 10 **Step** = 2) **Do**
  Write(i);
EndFor ;

Conclusion :
There are no precise rules to justify the choice of a type of loop (While, Repeat, For) but generally it depends on the desired processing:

- The **while** loop is the simplest and can be used in all situations.
- If in the algorithm/program we have a counter which increments/decrements by a precise step until reaching a final value, in this case, the **For** loop is best suited
- If the desired iteration requires the execution of a block at least once, in this case the **Repeat** loop is recommended;

5. Nested loops

Two loops are nested if one is contained within the instruction block of the other.
Some algorithms require nesting of loops, for example, we wish to display all the multiplication tables from 1 to 9.

- displaying the multiplication table of a number, for example the multiplication table of 7, requires a loop;
- in addition, we must display the multiplication table for each number between 1 and 9, which requires a second loop containing the first.

Example :
Consider the following algorithm which displays the multiplication tables from 1 to 9.

Analyse :

i x j = multiplication result
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63

Solution:

```
Algorithm multiplication_table;
 var
    i, j, r : integer;
 Begin
    For (i ←1 To 9) Do
        For (j ←1 To 9) Do
           r ← i * j ;
           Write (i, " x ", j, " = ", r);
        EndFor;
   EndFor;
end.
```

Exercises :

Write algorithms that calculates

- the sum S1 of odd numbers less than N : S1 = 1 + 3 + 4 + …..
- the sum S2 : S2 = 1 + 2 + 3 + … + N.
- the sum S3 : S3 = $1 + 2^2 + 3^2 + \ldots + N^2$.
- the power N of X : $X^N = X * X * \ldots * X$
- the factorial F of a number N : $F = N! = N * (N - 1) * (N - 2) * \ldots * 3 * 2 * 1$
- the sum S4 : S4 = 1 ! + 2 ! + 3 ! + … + N !

Chapter 5: Arrays and character strings

1. Arrays

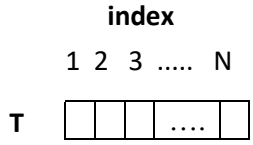
Arrays (or Tables) are data structures that allow manipulating multiple data with the same type.

We distinguish two types of arrays: one-dimensional arrays and two-dimensional arrays.

1.1. One-dimensional arrays

A. Definition

One-dimensional arrays are represented by a set of successive boxes identified by an index.



B. Declaration syntax:

Var

Array_Name : Array [1 .. Array_Size] of Data_Type;

To declare an array you must give it an identifier (name), a size and the type of data it will contain.

Example :

```
Var
  T : Array [1 .. 10] of integer;    // Array T of size 10 which contains integer type values
```

C. Arrays in C language :

Syntax :

Type Array_id [Array_Size] ;

Exemple :

int T[10] ;

D. Reading/Writing in one-dimensional arrays

- Read (Table_Id [index]); // allows reading a value from the keyboard and store it in the box indexed by the index value. Exp: **read(T[2]);**

- Write (Id_Tableau [index]); // allows displaying the contents of the box indexed by the value of the index. Exp: **write (T[1]);**

- Id_Table [index] ← expression; // allows assigning the expression evaluation result to the box identified by the index. Exp: **T[3] ← (A+B) div 2;**

E. Manipulating arrays

Generally, we use loops to manipulate arrays.

Exp: the following loop fills the table T with N values entered from the keyboard.

Algorithm	**Program in C**
For (i←1 To N)Do read (T[i]) ;	**For**(i = 0 ; i < N ; i++) Scanf(''%d'', &T[i]);

Exercise1 :
Write an algorithm that allows filling a table with 10 real values, and then calculate the sum and average of the values filled.

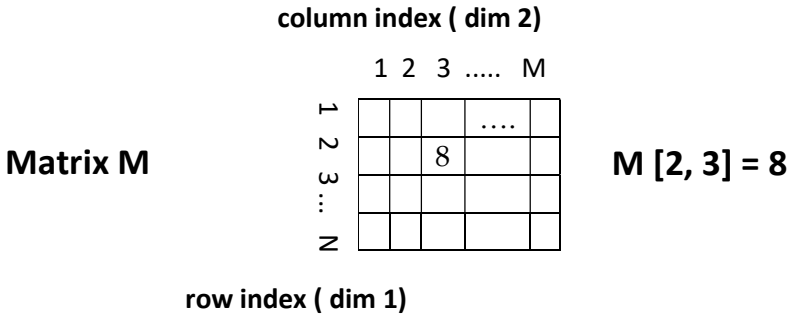
Exercise2 :
Write an algorithm that fills a table with 10 notes, and then calculates the number of notes greater than 10.

1.2. Two-dimensional arrays (matrix)

A. Definition

A Matrix is a data structure represented in the form of two-dimensional arrays, which allow manipulating data of the same type.

In programming, matrix boxes are identified by row numbers (index) and column numbers (index), which represent dimension 1 and dimension 2 respectively.



Note :
If the number of rows is equal to the number of columns (equal to N), we say that the matrix is square of size (or order) N.

B. Declaration

Var
  Matrix_name : Array [1 .. row_size, 1 .. column_size] of Type;

Example :
VAR
  M : Array [1 .. 6, 1.. 4] of integer ; // Matrix M of 6 rows and 4 columns with integer values

C. Declaration using C language

Type Array_id [row_size] [column_size] ;

Exemple :

int M[6][4] ;

D. Reading/Writing in Matrix

- Read (Matrix_id [row_index, column_index]); // allows to read a value from the keyboard and store it in the box indexed by the values of the row and column.
  Exp: Read(M[2, 3]);

- Write (Matrix_id [row_index, column_index]); // allows to display the contents of the box indexed by the values of the row and column.
  Exp: Write (M[1, 1]);

- Matrix_id [row_index, column_index] ← expression; // allows to assign the expression evaluation result to the box identified by the values of the row and column.
  Exp: M [3, 1] ← (A+B) mod 2;

E. Manipulation of Matrix

In general, we use two nested loops to manipulate a matrix.

Exp: the following loops fills a matrix of dimensions (N x M) with real type values entered on the keyboard.

Algorithm	Langage C

For (i←1 To N) **Do**
  For (j←1 To M) **Do**
  Read (M [i , j]) ;
	For(i = 0 ; i < N ; i++)
	  For(j = 0 ; j < M ; j++)
	  Scanf(''%f'', &M [i] [j]);

Exercise1:
Write an algorithm that fills a matrix with real type values, and then calculates the sum and average of the values in the matrix.

Exercise2:
Write a program that fills an N-dimensional square matrix with integer values. Then fill the first diagonal with 0s.

Exercise 3:
Write an algorithm that allows to fill an integer array T of size N and then search a value X in T.

Exercise 4:
Write an algorithm that allows filling an array with 10 notes, and then calculating the number of notes greater than 10.

Exercise 5:
Write an algorithm that allows entering the notes of 10 students who have 7 modules in a matrix M, then calculate the average of each student and store them in a table T. We assume that all the modules have the same coefficient.

Matrix of notes

Student\Notes	Module 1					Module 7
Student 1						
……						
student 10						

Array of average notes

Av 1	Av 2					Av 10

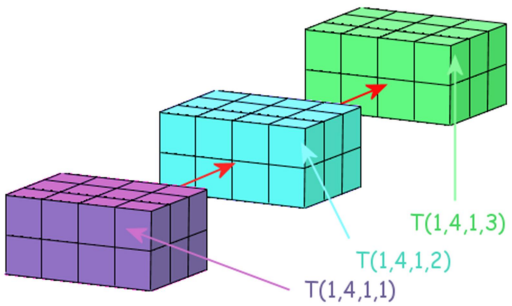
Exercise 6 :
Write an algorithm that finds the number of occurrences of X in a matrix.

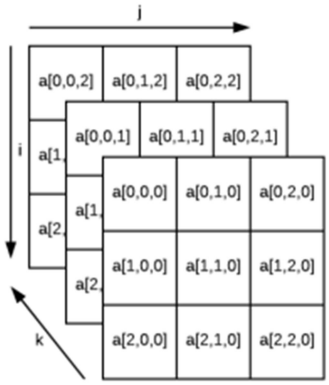
1.3. Multidimensional Arrays

An N-dimensional array is declared by specifying the number and type of values it will contain.

Exemples :
T1 : array [1..100, 1..100, 1…300] of integer;
T2 : array [1..10, 1..47, 1..12, 1..7] of real ;

The first is a three-dimensional array and the second is a four-dimensional array.





The reference of an element in an N-dimensional array is done in the same way as for a one or two-dimensional array, by specifying the index of each dimension.

2. Character strings

2.1. Definition

Strings are finite sequences of characters. The number of characters making the string is the length of this string. A zero-length string does not include any characters which is called an Empty string. In algorithms, a character string is designated between two Quotation mark " ".

Example :

- The string "Algo" constituted of the characters 'A', 'l', 'g', 'o'.
- The string "123" is not the numerical quantity 123.
- The string "" represents an empty string.
- The string " " is a character string containing only one character which is the space (do not confuse with the empty string).

2.2. Declaration

Syntax in algorithms :

var

id : string [length] of characters; **OR** id : string [length];

Syntax in C language:

There is no special type for strings in C language. A string is declared as a one-dimensional array of char, whose end is indicated by the character '\0'. The size of the string is equal to the maximum length of the string plus one so that we can store the character '\0' denoting the end.

Syntax :

char id [length] ;

Example :

Algorithm	Program in C
var ch : string [10] ;	char ch [10]; // maximum string length is 9

2.3. Initialization of a character string

To initialize a string, we declare a charter string identifier and assign its initial values.

Example :

In algorithm:

Var

ch : string [15] ;

ch ← ''Hello'' ;

In C language:

char ch[ ] = { 'H','e','l','l','o', '\0'}; // allocate 6 boxes (6 bytes)

// Using a literal string to initialize an array of unspecified size saves from worrying about the NULL character ('\0') since it is part of the literal string.

char ch[ ] = ''Hello'' ;

// The computer automatically reserves the number of bytes needed for the string, i.e.: the number of characters + 1 (here: 6 bytes).

```
char ch[6] = ''Hello'' ;
```

// If we specify the size of your array, we have to make sure that it has enough space to accommodate the entire string, that is to say the characters that compose it and the NULL character.

2.4. Access to the elements of a String

Access to an element of a string can be done in the same way as accessing an element of an array. Example: For the previous ch string, we have:

```
ch [0] = 'H' ; ch [1] = 'e' ; ch [2] = 'l' ; ch [3] = 'l' ; ch [4] = 'o' ;
```

2.5. Reading and displaying strings

In Algorithms, a character string is read (displayed) globally (at once) and not character by character.

In Algorithms:

```
Read (ch);
Write (ch);
```

In C program:

We can read (display) a character string in two ways:

- First way: scanf ("%s", & ch); printf("%s", ch);
- Second way: gets (ch); puts(ch);

// The difference between scanf and gets is that scanf only brings the text entered before the first blank (space) into the variable to read. However, gets brings all text entered text until to the carriage return (enter command).

Exercise :

Write an algorithm that allows to read a string, then check if the string contains the character 'e'.

Exercise :

Write an algorithm that allows to read two character strings ch1 and ch2, then check if the string ch2 is included in the string ch1.

2.6. Comparison of character strings

In a computer system, each character is associated with a numerical value: its ASCII code (American Standard Code for Information Interchange). All the codes are listed in a table called "ASCII code table". When we store a character in memory (in a variable), we memorize its ASCII code.

To compare two character strings, we compare the characters of the same rank in the two strings starting with the first character of each string (the first character of the first string is compared to the first character of the second string, the second character of the first string is compared to the second character of the second string, and so on…).

Examples: Comparing two strings

• "baobab" < "sport" because the ASCII code of b (98 in base 10) is lower than the ASCII code of 's' (115 in base 10).

• "baobab" > "banana" because the ASCII code of 'o' (111) is greater than the ASCII code of 'n' (110). The comparison cannot be made on the first two characters because they are identical.

• "333" > "1230" because the ASCII code of '3' (51) is greater than the ASCII code of '1' (49). Please note, here it is not numerical values that are compared, but rather characters.

• "333" < "3330" because the second string has a length greater than that of the first (the comparison cannot be made on the first three characters because they are identical).

• "Baobab" < "baobab" because the ASCII code of 'b' (98) is greater than the ASCII code of 'B' (66).

Exercise 1:

Write an algorithm that takes an uppercase string and converts it to lowercase.
Note :
the ASCII code for lowercases alphabetic characters are between 97 (a) and 122 (z).
the ASCII code for uppercases alphabetic characters are between 65 (A) and 90 (Z).

Exercise 2:

Consider a *Length (string)* function which returns the length of a character string.

Exanmple :

ch ← '' Hello '' ;
write (length (ch)) ; // display 5.

Write an algorithm that allows reading two strings of characters and then compare them if they are similar.

Solution :

```
Algorithm comparison;
Var
    ch1, ch2: string [10];
    i, j, k: integer;
Begin
 read (ch1, ch2);
 j ←length(ch1);
 k← length (ch2);
 i←1 ;
 While ( ( ch1[i] = ch2[i] ) AND (i < j) AND (k=j) ) Do
      i++;
 EndWhile ;
 If ( i = j )   Then
    write("ch1 = ch2");
 else
     write("ch1 ≠ ch2");
 endif ;
end.
```

2.7. Character string manipulation functions in C language

The <string.h> library contains several functions for manipulating character strings. Among them :

- The strlen(ch) function: provides the length of a string as a result.
- The strcat(ch1,ch2) function: copies the second string ch2 after the first string ch1.
- The strcmp(ch1, ch2) function: compares two strings and provides a positive integer value if ch1>ch2, zero if ch1=ch2 and negative if ch1<ch2.
- The strcpy(destination, source) function: copies the source string into the destination string address.

Chapter 6: Custom types

In addition to predefined types (standard), the programmer can define new types. In this course we are mainly interested in types: Enumeration and Records.

1. Enumerations

An enumeration is a type whose area of values is defined by the programmer.

- The months of the year (January, February ...);
- Playing cards names (AS, King...);
- Car marks (Peugeot, Renault, Fiat, ...);
- Civil status indications (single, married, divorced ...).

Syntax of declaration of a listed type:

Type Type_name = (Val1, Val2, ……., Valn);

Declaration of a variable of an enumerated type:

Var

variable_name : name_ type ;

Examples:

Type

day = (Saturday, Sunday, Monday, Tuesday, Wednesday, Thursday, Friday);

month = (January, February, March, April, May, June, July, August, September, October, November, December) ;

Var

D1, D2: Day;

M: month;

- Variables D1 and D2 can only take one of the values: Saturday... Friday.
- The variable M can only take one of the values: January... December

Constants of an enumeration are linked by an order-relation defined by the position of values in an enumeration. Then, the order in which identifiers are listed is significant. Examples: Saturday<Monday and December> January.

The names attributed to the various constants of an enumeration cannot be reused.

```
Var
  Saturday: integer; // error !!!
```

Some functions can be used to manipulate enumerated types:

- Ord (x): This function returns a positive integer corresponding to the rank of x element in the enumeration.
- Succ (x): This function provides the constant which immediately follows the value of X in the enumeration. The successor of the last value is not defined.
- Pred (x): This function provides the constant which immediately precedes the value of X in the list. The first value predecessor is not defined.

Examples:

- Ord (Saturday) = 1, ord (Sunday) = 2, .. ord (Friday) = 7.
- Succ(Saturday) = Sunday, Succ (Sunday) = Monday,…, Succ (Friday) =? (is not defined).
- Pred (Friday) = Thursday, pred (Thursday) = Wednesday,…, pred (Saturday) =? (is not defined).

Syntax of enumeration in C language:

To declare such a type, we start with the **Enum** keyword. As following:

Enum Name_ Type {Val1, Val2, …, Valn};

Example:

Enum Day {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};

Declaration of a variable of an enumerated type:

Enum Name_ type Name_variable;

Note:

The C language considers the values of the types enumerated as integer constants, converting them in the order in which they were listed during the declaration from 0.

Example :

```
#include <stdio.h>

Enum day {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

int main()
{
   enum day date;
   date = Tuesday;
   Printf ("The %d day of the week is : ", date+1);
   return 0;
}
```

2. The records

1.3. Definition

A record is a data structure allowing a set of data of different types to be grouped with the same and single object.

A record (also called a structure) is made up of components called fields.

Each field is identified by a type and a name which allows direct access to it.

1.4. Declaration

```
Type id_record = Record
                     Id_Field1: Type1 ;
                     Id_Field2: Type2 ;
                     …
                     Id_FieldN: TypeN ;
                 End;
```

- Id_Field1, Id_Field2 … Id_FieldN: Are the identifiers of the fields of the record.
- Type1, Type2, …, TypeN : Are the types associated with the fields.
- Once the record type is defined, you can declare variables of this type.

Syntax :

```
Var
    Id_variable : id_record ;
```

Example :

Consider the information concerning a student: name, age, email, baccalaureate note, can be represented using a record as follows:

```
Type student = Record
                  name: string [N];
                  age: int ;
                  email: string [N];
                  bac: Real;
               end;
var
    S1, S2, S3: Student;
```

A record can be represented by a set of boxes. These boxes can be different sizes, because the types of a record are not necessarily the same as for a table.

	Name	age	email	bac
S1	''Biskri Ali''	19	Biskri.ali@gmail.com	14.25

1.5. Access to a field of a record

The fields of a record are accessible using the variable identifier and the field name separated by a dot (.)

Syntax:

Id_Variable.id_field

Example 1:

S1.age
S2.email

Since the fields of a record correspond to a consecutive space of bytes, therefore they play the role of variables. They can thus be used in assignment, reading, writing, etc. actions.

Example 2:

```
S1.age ← 21 ;
read (S1.name) ;
write (S1.email) ;
```

1.6. Records in C language

Syntaxe :

```
Typedef struct {
                    Type1  Id_Field1;
                    Type2  Id_Field2;
                     ...
                    TypeN  Id_FieldN;
                }id_record;
```

Example :

```
Typedef struct {
                    Char name [10] ;
                    Int  age;
                    Char email  [10] ;
                    Float bac ;
                } Student ;
```

Example :

```
#include <stdio.h>

typedef struct {
                    char name[20];
                    int age;
                  } person;

int main()
{
   person p;
   gets(p.name);
   scanf("%d",& p.age);
   printf("\n  The name is : ");
   puts(p.name);
   printf("\n The age is : %d  ", p.age );

  return 0;
}
```

1.7. Case of nested structures

A record can be nested in a table or record types. The notation used to select fields remains the same (Use of point).

Array of records:
It is possible to declare an array whose elements are of record type.

```
Type  id_record   =  Record
                          Id_Field1: Type1 ;
                          Id_Field2: Type2 ;
                          …
                          Id_FieldN: TypeN ;
                    End;
 Var
    id_array : array [ 1 .. N ] of id_record ;
```

1.8. Access to elements

We first access the table box, using the brackets [ ], then we access the field using the dot symbol (.)

Example:

Id_array [i] . id_Field;

1.9. Manipulating arrays with record type

- Read(Tab[2].age); // save the value in the age field of the 2nd element in the array.
- Tab[3]. bac ← 13.50; // assign a value 13.50 to the field bac in the 3rd box of the table.
- Write(Tab[1].name); // display the name in the 1st box of the table.

Exercise:

Using records and arrays structures, write an algorithm which allows:

1- Creates and fill a data base for students (name, age, email, phone number).

2- Calculate the number of students with age superior of 28

References

[1] J. Courtin et al. ***Initiation à l'algorithmique et aux structures de données. 2ème Edition*** DUNOD. 1998.

[2] M. Divay. Algorithmes et structures de données génériques - 2ème édition. Edition Dunod. 2004.

[3] M. C. Belaid. ***Algorithmique Programmation en Pascal***. Cours, Exercices, Travaux Pratiques, Corrigés, Bouira-Algérie. Pages Bleues Internationales. 2008.

[4] J. Tisseau***. Initiation à l'algorithmique***. Presse École nationale d'ingénieurs de Brest, 2009.

[5] T. H. Cormen, Charles E. Leiserson, Ronald L. Rivest. ***Algorithmique - 3ème édition –*** Cours avec 957 exercices et 158 problèmes Broché. Dunod. 2010.

[6] T. H. Cormen. ***Algorithmes Notions de base Collection : Sciences Sup.*** Dunod. 2013.

[7] D. Brethet, V. Labatut. ***Algorithmiques et programmation en language C***. Support de cours Vol 1. Université Galatasaray. Version 2. 2019.

[8] B. Salim. ***Séries d'exercices de travaux dirigés et pratiques en algorithmique***. Université de Biskra. 2021.

[9] M. Kara. ***Algorithmiques et Structures de données 1.*** Support de cours. Université de Jijel. 2021.